



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2020-0080165
(43) 공개일자 2020년07월06일

(51) 국제특허분류(Int. Cl.)

G06F 3/06 (2006.01)

(52) CPC특허분류

G06F 3/061 (2013.01)

G06F 3/064 (2013.01)

(21) 출원번호 10-2019-0170007

(22) 출원일자 2019년12월18일

심사청구일자 2019년12월18일

(30) 우선권주장

1020180169842 2018년12월26일 대한민국(KR)

(71) 출원인

중앙대학교 산학협력단

서울특별시 동작구 흑석로 84 (흑석동)

(72) 발명자

박상오

서울특별시 동작구 상도로47바길 40, 101동 202호(상도동, 상도패리스)

김성조

서울특별시 강남구 압구정로 201, 83동 901호(압구정동, 현대아파트)

(뒷면에 계속)

(74) 대리인

오위환, 나성곤, 정기택

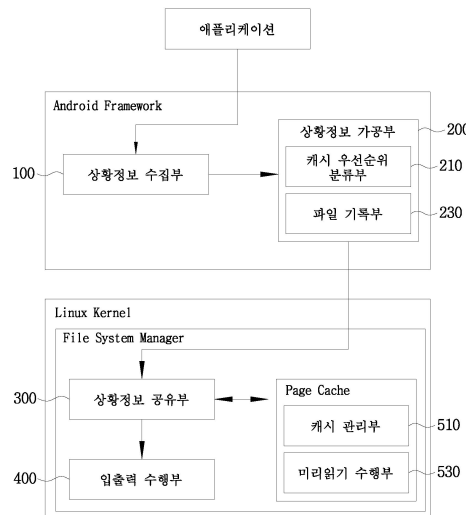
전체 청구항 수 : 총 16 항

(54) 발명의 명칭 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템 및 방법

(57) 요약

본 발명은 사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집하는 상황정보 수집부, 상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일을 순차적으로 애플리케이션 저장소에 기록하여 상기 상황정보를 가공하는 상황정보 가공부, 상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행하는 상황정보 공유부, 상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행하는 입출력 수행부를 포함하여 리눅스 커널의 입출력 스택의 각 레이어 간의 정보 공유를 통해 각 레이어의 구성요소 간 협동적 동작을 구현할 수 있는 파일 입출력 관리 기술을 제공할 수 있는 효과를 가진다.

대표도 - 도2



(52) CPC특허분류

G06F 3/0656 (2013.01)

G06F 2003/0691 (2013.01)

(72) 발명자

이재환

서울특별시 동작구 상도로 48마길 20 에콰빌 306

남상혁

서울특별시 동작구 상도로41길 24 해밀원룸 202호

박승우

서울특별시 동작구 흑석로 8, 405

곽수환

경기도 안양시 동안구 평촌대로427번길 60-26 넓은
들빌라

김경찬

경기도 과천시 향촌6길 10-12, 102호(별양동)

이 발명을 지원한 국가연구개발사업

과제고유번호 1711093476

부처명 과학기술정보통신부

연구관리전문기관 정보통신기획평가원

연구사업명 정보통신기술인력양성(R&D)

연구과제명 블록체인 비즈니스 서비스 기술 개발 및 인력양성

기 여 율 1/2

주관기관 중앙대학교산학협력단

연구기간 2019.04.01 ~ 2019.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호 1711090445

부처명 과학기술정보통신부

연구관리전문기관 한국연구재단

연구사업명 개인기초연구(과기정통부)(R&D)

연구과제명 포그 컴퓨팅 패러다임과 소프트웨어 정의 네트워크를활용한 자율 군집주행 지원 프레임워

크

기 여 율 1/2

주관기관 중앙대학교

연구기간 2019.03.01 ~ 2019.08.31

명세서

청구범위

청구항 1

사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집하는 상황정보 수집부;

상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일들이 순차적으로 접근된 패턴을 애플리케이션 저장소에 기록하여 상기 상황정보를 가공하는 상황정보 가공부;

상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행하는 상황정보 공유부;

상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행하는 입출력 수행부;를 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

청구항 2

제 1 항에 있어서, 상기 상황정보 수집부에서 수집되는 상황정보는,

상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 접근 횟수, 사용 시간에 기초한 사용빈도 점수를 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

청구항 3

제 2 항에 있어서, 상기 상황정보 가공부는,

상기 수집된 상황정보에 기초한 캐시 적중률 점수에 따라 상기 파일에 대한 캐시 우선순위를 분류하는 캐시 우선순위 분류부; 및

상기 분류된 캐시 우선순위 중 기 설정된 우선순위 캐시에 해당하는 파일을 상기 애플리케이션 저장소에 기록하는 파일 기록부;를 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템

청구항 4

제 3 항에 있어서, 상기 캐시 우선순위 분류부는,

상기 캐시 적중률 점수가 상위 1/3 이상이면 높은 우선순위 캐시(High priority cache)로 분류하고,

상기 캐시 적중률 점수가 상위 2/3 이상 1/3 이하이면 표준 우선순위 캐시(Normal priority cache)로 분류하고,

상기 캐시 적중률 점수가 2/3 미만이면 낮은 우선순위 캐시(Low priority cache)로 분류하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

청구항 5

제 4 항에 있어서,

상기 높은 우선순위 캐시는 우선순위 큐(Priority queue)로 관리하고,

상기 표준 우선순위 캐시는 LRU 큐로 관리하고,

상기 낮은 우선순위 캐시는 FIFO 큐로 관리하는 캐시 관리부;를 더 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

청구항 6

제 4 항에 있어서,

상기 높은 우선순위 캐시(High priority cache)에 해당하는 파일을 단일파일과 다중파일로 구분하여 미리읽기(Readahead)를 각각 수행하는 미리읽기 수행부;를 더 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

청구항 7

제 2 항에 있어서, 상기 사용빈도 점수는,

하기 수학적 식 1로 표현되는 애플리케이션 사용빈도 점수 산출식을 이용하여 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 사용빈도를 산출하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

<수학적 식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid})} + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}$$

(uid: 애플리케이션의 ID, $AUScore_{uid}$: 애플리케이션 uid의 사용빈도 점수, $WeeklyOpenCount_{uid}$: 7일간 애플리케이션 uid의 접근 횟수들의 집합, $WeeklyFGtime_{uid}$: 7일 간 애플리케이션 uid의 사용 시간들의 집합, $OpenCount_{uid}$: 일별 애플리케이션 uid의 접근 횟수, $FGtime_{uid}$: 일별 애플리케이션 uid의 사용 시간)

청구항 8

제 3 항에 있어서, 상기 캐시 적중률 점수는,

하기 수학적 식 2로 표현되는 캐시 적중률 점수 산출식을 이용하여 상기 수집된 상황정보에 기초한 상기 파일에 대한 캐시 적중률 점수를 산출하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템.

<수학적 식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

(inode: 파일의 inode ID, $CHScore_{inode}$: 캐시 적중률 점수, $ReadCacheHit$: 읽기 캐시 적중 수의 집합, $WriteCacheHit$: 쓰기 캐시 적중 수의 집합, $ReadCacheHit_{inode}$: 파일 inode의 읽기 캐시 적중 수, $WriteCacheHit_{inode}$: 파일 inode의 쓰기 캐시 적중 수)

청구항 9

상황정보 수집부에서 사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집하는 상황정보 수집단계;

상황정보 가공부에서 상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일을 순차적으로 애플리케이션 저장소에 기록하여 상기 상황정보를 가공하는 상황정보 가공단계;

상황정보 공유부에서 상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행하는 상황정보 공유단계;

입출력 수행부에서 상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행하는 입출력 수행단계;를 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

청구항 10

제 9 항에 있어서, 상기 상황정보 수집단계에서 수집되는 상황정보는,
상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 접근 횟수, 사용 시간에 기초한 사용빈도 점수를 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

청구항 11

제 10 항에 있어서, 상기 상황정보 가공단계는,
캐시 우선순위 분류부에서 상기 수집된 상황정보에 기초한 캐시 적중률 점수에 따라 상기 파일에 대한 캐시 우선순위를 분류하는 캐시 우선순위 분류단계; 및
파일 기록부에서 상기 분류된 캐시 우선순위 중 기 설정된 우선순위 캐시에 해당하는 파일을 상기 애플리케이션 저장소에 기록하는 파일 기록단계;를 포함하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

청구항 12

제 11 항에 있어서 상기 캐시 우선순위 분류단계는,
상기 캐시 적중률 점수가 상위 1/3 이상이면 높은 우선순위 캐시(High priority cache)로 분류하고,
상기 캐시 적중률 점수가 상위 2/3 이상 1/3 이하이면 표준 우선순위 캐시(Normal priority cache)로 분류하고,
상기 캐시 적중률 점수가 2/3 미만이면 낮은 우선순위 캐시(Low priority cache)로 분류하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

청구항 13

제 12 항에 있어서, 캐시 관리부에서,
상기 높은 우선순위 캐시는 우선순위 큐(Priority queue)로 관리하고,
상기 표준 우선순위 캐시는 LRU 큐로 관리하고,
상기 낮은 우선순위 캐시는 FIFO 큐로 관리하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

청구항 14

제 12 항에 있어서, 미리읽기 수행부에서,
상기 높은 우선순위 캐시(High priority cache)에 해당하는 파일을 단일파일과 다중파일로 구분하여 미리읽기(Readahead)를 각각 수행하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

청구항 15

제 10 항에 있어서, 상기 사용빈도 점수는,
하기 수학적 식 1로 표현되는 애플리케이션 사용빈도 점수 산출식을 이용하여 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 사용빈도를 산출하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

<수학적 식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid})} + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}$$

(uid: 애플리케이션의 ID, $AUScore_{uid}$: 애플리케이션 uid의 사용빈도 점수, $WeeklyOpenCount_{ui}$: 7일간 애플리케이션 uid의 접근 횟수들의 집합, $WeeklyFGtime_{uid}$: 7일 간 애플리케이션 uid의 사용 시간들의 집합, $OpenCount_{uid}$: 일별 애플리케이션 uid의 접근 횟수, $FGtime_{uid}$: 일별 애플리케이션 uid의 사용 시간)

청구항 16

제 11 항에 있어서, 상기 캐시 적중률 점수는,

하기 수학적 식 2로 표현되는 캐시 적중률 점수 산출식을 이용하여 상기 수집된 상황정보에 기초한 상기 파일에 대한 캐시 적중률 점수를 산출하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법.

<수학적 식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

(inode: 파일의 inode ID, $CHScore_{inode}$: 캐시 적중률 점수, $ReadCacheHit$: 읽기 캐시 적중 수의 집합, $WriteCacheHit$: 쓰기 캐시 적중 수의 집합, $ReadCacheHit_{inode}$: 파일 inode의 읽기 캐시 적중 수, $WriteCacheHit_{inode}$: 파일 inode의 쓰기 캐시 적중 수)

발명의 설명

기술 분야

[0001] 본 발명은 모바일 기기에서의 파일 입출력 관리 기술에 관한 것으로서, 보다 상세하게는 입출력 스택의 각 레이어 간 정보 공유를 통해 효율적인 파일의 입출력을 관리할 수 있도록 하는 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템 및 방법에 관한 것이다.

배경 기술

[0002] 모바일 장치에서의 애플리케이션 동작 간의 실행시간을 줄여 애플리케이션의 반응성을 높이고, 이를 통해 사용자의 사용경험을 높이기 위해서 파일 입출력 스택 및 파일시스템의 성능향상을 위한 많은 연구들이 이루어지고 있다.

[0003] 그 중 리눅스 커널의 입출력 스케줄러 관련 연구들이 많이 진행되고 있는데, 이러한 연구들은 리눅스 커널 입출력 스택의 하부에 있는 입출력 스케줄러만을 대상으로 개선이 이루어졌기 때문에, 입출력 스택의 VFS 레이어나 유저 애플리케이션과 같은 상위 레이어에서 발생할 수 있는 비효율성에 대해서는 하위 레이어에서 얻을 수 있는 정보가 한정되어 있기 때문에 개선의 한계가 명확한 한계점을 안고 있다.

[0004] 한편, 모바일 장치에서 주 저장장치로 쓰이는 플래시 메모리는 입출력 특성이 하드디스크와 크게 다르기 때문에, 하드디스크를 고려하여 설계된 ext4 파일시스템 대신 플래시 메모리의 특성에 적합한 f2fs 파일시스템이 개발되었다.

[0005] f2fs 파일시스템은 NAND 기반 저장소의 특성을 활용하여 하드웨어 저장장치의 FTL 특성에 따라 segment, section, zone으로 구성된 디스크 레이아웃을 가지며, segment는 FTL의 접근단위(NAND Page), section은 FTL의 삭제단위(NAND Block)로 설정하며, zone은 기본적으로 section과 같은 크기로 설정되나, 사용자가 설정할 수 있도록 하여 FTL의 병렬성을 활용할 수 있도록 한다.

[0006] 또한, 특정 위치의 데이터를 덮어쓰기 위해서 먼저 삭제단위(NAND Block)만큼 삭제를 한 다음에 새 데이터를 써야 하는 NAND 플래시 메모리의 특성을 고려하여 데이터를 덮어 쓰는 것이 아닌, 새로운 데이터를 바뀐 부분만 기록하는 방식의 LFS(Log-structured file system)형식을 취한다. 또한, 주기적으로 조각난 파일을 모으는 방식을 취하여 하드웨어 저장장치에서 불필요한 데이터의 복제 혹은 삭제를 줄이도록 하여 스마트폰에 주로 쓰이는

NAND 기반 저장장치 장치에 적합하며 기존 ext4 파일 시스템보다 더 높은 성능을 보여준다.

- [0007] 다만 마찬가지로, 리눅스 입출력 스택 중 하나의 레이어인 파일시스템의 개선만 이루어졌기 때문에, 해당 레이어 상위 레이어와 하위 레이어에서 발생하는 비효율성에 대해서는 개선이 이루어지지 않은 한계점을 안고 있다.
- [0008] 기존 연구들은 운영체제 입출력 스택의 일부분을 개선하여 모바일 장치에서의 입출력 성능을 개선시키기 위한 연구들이었으나, 리눅스 입출력 스택 레이어 간 제한된 정보 교환으로 인한 레이어 간 비효율성에 대한 연구가 부족하고, 이를 해결하기 위한 스택 전반의 협업을 통한 개선에 대한 연구는 미흡한 한계점이 있다.
- [0009] 또한, 스마트폰에서 보여지는 애플리케이션의 특수한 실행 패턴을 활용한 개선보다는 순차 입출력, 랜덤 입출력 등 일반적인 입출력 특성만을 고려하여 연구가 진행된 바 있다.

선행기술문헌

특허문헌

- [0010] (특허문헌 0001) 미국공개특허공보 제2017-0052979호

발명의 내용

해결하려는 과제

- [0011] 이에 본 발명은 상기와 같은 종래의 제반 문제점을 해소하기 위해 제안된 것으로, 리눅스 커널의 입출력 스택의 각 레이어 간의 정보 공유를 통해 각 레이어의 구성요소 간 협동적 동작을 구현할 수 있는 파일 입출력 관리 기술을 제공하는 것을 목적으로 한다.
- [0012] 또한, 본 발명은 스마트폰의 애플리케이션 실행패턴을 활용하여 애플리케이션 실행에 걸리는 시간을 줄임으로써 사용자가 더 빠르게 애플리케이션을 사용할 수 있도록 하고, 향상된 사용자 경험을 제공할 수 있도록 하는 것을 목적으로 한다.
- [0013] 또한, 본 발명은 애플리케이션에 대한 파일의 상황정보를 가공 및 공유함으로써, 다른 모듈에서도 활용할 수 있도록 하는 것을 목적으로 한다.
- [0014] 또한, 본 발명은 캐시된 파일을 우선순위에 따라 분류함으로써, 캐시된 파일의 효율적인 관리를 도모하는 것을 목적으로 한다.
- [0015] 또한, 본 발명은 앞으로 필요할 수 있는 파일을 미리 메모리로 읽음으로써, 파일의 캐시 적중률을 높이는 것을 목적으로 한다.

과제의 해결 수단

- [0016] 상기와 같은 목적을 달성하기 위하여 본 발명의 기술적 사상에 의한 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템은, 사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집하는 상황정보 수집부, 상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일을 순차적으로 애플리케이션 저장소에 기록하여 상기 상황정보를 가공하는 상황정보 가공부, 상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행하는 상황정보 공유부, 상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행하는 입출력 수행부를 포함할 수 있다.
- [0017] 이 때 상황정보 수집부에서 수집되는 상황정보는 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 접근 횟수, 사용 시간에 기초한 사용빈도 점수를 포함할 수 있다.
- [0018] 사용빈도 점수는 하기 수학식 1로 표현되는 애플리케이션 사용빈도 점수 산출식을 이용하여 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 사용빈도를 산출할 수 있다.

[0019] <수학식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid})} + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}$$

[0020]

[0021] (uid : 애플리케이션의 ID, $AUScore_{uid}$: 애플리케이션 uid 의 사용빈도 점수, $WeeklyOpenCount_{ui}$: 7일간 애플리케이션 uid 의 접근 횟수들의 집합, $WeeklyFGtime_{uid}$: 7일 간 애플리케이션 uid 의 사용 시간들의 집합, $OpenCount_{uid}$: 일별 애플리케이션 uid 의 접근 횟수, $FGtime_{uid}$: 일별 애플리케이션 uid 의 사용 시간)

[0022] 상기 상황정보 가공부는 상기 수집된 상황정보에 기초한 캐시 적중률 점수에 따라 상기 파일에 대한 캐시 우선순위를 분류하는 캐시 우선순위 분류부, 상기 분류된 캐시 우선순위 중 기 설정된 우선순위 캐시에 해당하는 파일을 상기 애플리케이션 저장소에 기록하는 파일 기록부를 포함할 수 있다.

[0023] 상기 캐시 적중률 점수는 하기 수학식 2로 표현되는 캐시 적중률 점수 산출식을 이용하여 상기 수집된 상황정보에 기초한 상기 파일에 대한 캐시 적중률 점수를 산출할 수 있다.

[0024] <수학식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

[0025]

[0026] ($inode$: 파일의 inode ID, $CHScore_{inode}$: 캐시 적중률 점수, $ReadCacheHit$: 읽기 캐시 적중 수의 집합, $WriteCacheHit$: 쓰기 캐시 적중 수의 집합, $ReadCacheHit_{inode}$: 파일 inode의 읽기 캐시 적중 수, $WritecacheHit_{inode}$: 파일 inode의 쓰기 캐시 적중 수)

[0027] 상기 캐시 우선순위 분류부는 상기 캐시 적중률 점수가 상위 1/3 이상이면 높은 우선순위 캐시(High priority cache)로 분류하고, 상기 캐시 적중률 점수가 상위 2/3 이상 1/3 미만이면 표준 우선순위 캐시(Normal priority cache)로 분류하고, 상기 캐시 적중률 점수가 2/3 미만이면 낮은 우선순위 캐시(Low priority cache)로 분류할 수 있다.

[0028] 한편, 본 발명의 기술적 사상에 의한 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템은, 상기 높은 우선순위 캐시는 우선순위 큐(Priority queue)로 관리하고, 상기 표준 우선순위 캐시는 LRU 큐로 관리하고, 상기 낮은 우선순위 캐시는 FIFO 큐로 관리하는 캐시 관리부를 더 포함할 수 있다.

[0029] 또한, 상기 높은 우선순위 캐시(High priority cache)에 해당하는 파일을 단일파일과 다중파일로 구분하여 미리 읽기(Readahead)를 각각 수행하는 미리읽기 수행부를 더 포함할 수 있다.

[0030] 상기과 같은 목적을 달성하기 위하여 본 발명의 기술적 사상에 의한 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법은, 상황정보 수집부에서 사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집하는 상황정보 수집단계, 상황정보 가공부에서 상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일을 순차적으로 애플리케이션 저장소에 기록하여 상기 상황정보를 가공하는 상황정보 가공단계, 상황정보 공유부에서 상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행하는 상황정보 공유단계, 입출력 수행부에서 상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행하는 입출력 수행단계를 포함할 수 있다.

[0031] 이 때 상기 상황정보 수집단계에서 수집되는 상황정보는 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 접근 횟수, 사용 시간에 기초한 사용빈도 점수를 포함할 수 있다.

[0032] 상기 사용빈도 점수는 하기 수학식 1로 표현되는 애플리케이션 사용빈도 점수 산출식을 이용하여 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 사용빈도를 산출할 수 있다.

[0033] <수학식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid}) + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}}$$

[0034]

[0035] (uid: 애플리케이션의 ID, $AUScore_{uid}$: 애플리케이션 uid의 사용빈도 점수, $WeeklyOpenCount_{ui}$: 7일 간 애플리케이션 uid의 접근 횟수들의 집합, $WeeklyFGtime_{uid}$: 7일 간 애플리케이션 uid의 사용 시간들의 집합, $OpenCount_{uid}$: 일별 애플리케이션 uid의 접근 횟수, $FGtime_{uid}$: 일별 애플리케이션 uid의 사용 시간)

[0036] 상기 상황정보 가공단계는 캐시 우선순위 분류부에서 상기 수집된 상황정보에 기초한 캐시 적중률 점수에 따라 상기 파일에 대한 캐시 우선순위를 분류하는 캐시 우선순위 분류단계, 파일 기록부에서 상기 분류된 캐시 우선순위 중 기 설정된 우선순위 캐시에 해당하는 파일을 상기 애플리케이션 저장소에 기록하는 파일 기록단계를 포함할 수 있다.

[0037] 상기 캐시 적중률 점수는 하기 수학식 2로 표현되는 캐시 적중률 점수 산출식을 이용하여 상기 수집된 상황정보에 기초한 상기 파일에 대한 캐시 적중률 점수를 산출할 수 있다.

[0038] <수학식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

[0039]

[0040] (inode: 파일의 inode ID, $CHScore_{inode}$: 캐시 적중률 점수, $ReadCacheHit$: 읽기 캐시 적중 수의 집합, $WriteCacheHit$: 쓰기 캐시 적중 수의 집합, $ReadCacheHit_{inode}$: 파일 inode의 읽기 캐시 적중 수, $WritecacheHit_{inode}$: 파일 inode의 쓰기 캐시 적중 수)

[0041] 상기 캐시 우선순위 분류단계는 상기 캐시 적중률 점수가 상위 1/3 이상이면 높은 우선순위 캐시(High priority cache)로 분류하고, 상기 캐시 적중률 점수가 상위 2/3 이상 1/3 미만이면 표준 우선순위 캐시(Normal priority cache)로 분류하고, 상기 캐시 적중률 점수가 2/3 미만이면 낮은 우선순위 캐시(Low priority cache)로 분류할 수 있다.

[0042] 캐시 관리부에서 상기 높은 우선순위 캐시는 우선순위 큐(Priority queue)로 관리하고, 상기 표준 우선순위 캐시는 LRU 큐로 관리하고, 상기 낮은 우선순위 캐시는 FIFO 큐로 관리할 수 있다.

[0043] 미리읽기 수행부에서 상기 높은 우선순위 캐시(High priority cache)에 해당하는 파일을 단일파일과 다중파일로 구분하여 미리읽기(Readahead)를 각각 수행할 수 있다.

발명의 효과

[0044] 본 발명에 의한 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템 및 방법에 따르면,

[0045] 첫째, 리눅스 커널의 입출력 스택의 각 레이어 간의 정보 공유를 통해 각 레이어의 구성요소 간 협동적 동작을 구현할 수 있는 파일 입출력 관리 기술을 제공할 수 있는 효과를 가진다.

[0046] 둘째, 스마트폰의 애플리케이션 실행패턴을 활용하여 애플리케이션 실행에 걸리는 시간을 줄임으로써 사용자가 더 빠르게 애플리케이션을 사용할 수 있도록 하고, 향상된 사용자 경험을 제공할 수 있는 효과를 가진다.

[0047] 셋째, 애플리케이션에 대한 파일의 상황정보를 가공 및 공유함으로써, 다른 모듈에서도 활용할 수 있는 효과를

가진다.

[0048] 넷째, 캐시된 파일을 우선순위에 따라 분류함으로써, 캐시된 파일을 효율적으로 관리할 수 있는 효과를 가진다.

[0049] 다섯째, 앞으로 필요할 수 있는 파일을 미리 메모리로 읽음으로써, 파일의 캐시 적중률을 높일 수 있는 효과를 가진다.

도면의 간단한 설명

[0050] 도 1은 일반적인 리눅스 커널 입출력 스택을 나타낸 구성도.

도 2는 본 발명의 실시예로서, 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템을 나타낸 구성도.

도 3은 본 발명의 실시예로서, 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법을 나타낸 순서도.

도 4는 본 발명의 실시예에 따라 본 발명의 시스템을 안드로이드 운영체제에 적용한 구성도를 나타낸 도면.

도 5는 본 발명의 실시예에 따른 본 발명의 성능 평가 중, 애플리케이션 실행시간 테스트 결과를 나타낸 도면.

도 6은 본 발명의 실시예에 따른 본 발명의 성능 평가 중, 캐시 적중률 테스트 결과를 나타낸 도면.

도 7은 본 발명의 실시예에 따른 본 발명의 성능 평가 중, 읽기 속도 테스트 결과를 나타낸 도면.

발명을 실시하기 위한 구체적인 내용

[0051] 첨부한 도면을 참조하여 본 발명의 실시예들에 의한 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템 및 방법에 대하여 상세히 설명한다. 본 발명은 다양한 변경을 가할 수 있고 여러 가지 형태를 가질 수 있는바, 특정 실시예들을 도면에 예시하고 본문에 상세하게 설명하고자 한다. 그러나 이는 본 발명을 특정한 개시 형태에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다. 각 도면을 설명하면서 유사한 참조부호를 유사한 구성요소에 대해 사용하였다.

[0052] 또한, 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가지고 있다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥 상 가지는 의미와 일치하는 의미를 가지는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0054] 도 1은 일반적인 리눅스 커널 입출력 스택을 나타낸 구성도이다.

[0055] 도 1을 참조하면, 일반적인 리눅스 커널 입출력 스택은 VFS(Virtual File System) 레이어, 블록 레이어(Block Layer), 디바이스 드라이버(Device Drivers) 등을 포함하여, 상위 애플리케이션(Applications)이 하드디스크나 플래시 저장소와 같은 보조 저장장치에 데이터를 읽거나 쓸 수 있도록 지원할 수 있다.

[0056] (1) VFS(Virtual File System) 레이어

[0057] 리눅스 커널 입출력 스택에서 VFS(Virtual File System) 레이어는 블록저장소 기반의 파일시스템(Block-based File System), 네트워크 기반 파일시스템(Network File System), 동적으로 생성되는 데이터를 파일의 형태로 제공하는 수도 파일시스템(Pseudo File System), 장치 드라이버나 메인 메모리 공간을 제공하는 특수 파일시스템(Special File system) 등 여러가지 형태로 제공될 수 있는 파일 시스템들에서 공통적으로 사용하는 기능들을 정의하고, 통합적으로 관리 및 사용할 수 있도록 추상화 레이어를 제공하는 역할을 한다.

[0058] (2) 파일 시스템(File System)

[0059] 리눅스 커널 입출력 스택에서 파일 시스템(File System)은 데이터를 관리하기 위하여 데이터 저장 체계와 접근 방법 등을 제공하는 시스템이라 할 수 있다. 리눅스에서는 이 데이터를 파일이라는 단위로 처리를 할 수 있다. 리눅스는 다양한 파일시스템들을 지원하며, 대표적으로 하드디스크나 SSD와 같은 블록 저장장치에 사용되는 Ext3,4와 같은 Ext 계열 파일시스템과, 컨트롤러가 없는 NAND 메모리와 같은 MTD(Memory Technology Device)

저장장치에 사용되는 JFFS2, YAFFS와 같은 파일시스템들을 제공할 수 있다.

[0060] (3) 페이지 캐시(Page Cache)

[0061] 리눅스 커널 입출력 스택에서 페이지 캐시(Page Cache)는 보조 저장장치에 대한 캐시를 제공하며, 모든 입출력은 페이지 캐시를 거쳐서 이루어질 수 있다. 페이지 캐시는 보조 저장장치에서 읽어온 데이터와 보조 저장장치에 쓰야할 데이터를 모두 페이지라는 단위로 관리를 할 수 있다. 이 때 이 페이지들은 메인 메모리에서 현재 사용 중이 아닌 공간에 할당을 하도록 하여 보조 저장장치 입출력에 대한 캐시로서 사용되어 매번 느린 보조 저장장치에서 데이터를 읽어오지 않고 빠른 주 메모리에서 바로 캐시 된 데이터를 읽어올 수 있도록 하여 입출력 성능을 높일 수 있도록 할 수 있다.

[0062] (4) 블록 레이어(Block Layer)

[0063] 리눅스 커널 입출력 스택에서 블록 레이어(Block Layer)는 하드웨어 저장장치에 직접 접근하기 위한 레이어로, 하드웨어 저장장치에서 임의의 데이터를 임의의 위치에서 읽어오거나 쓸 수 있다. 이 때, 리눅스 블록 레이어는 하드웨어 저장장치로의 입출력의 단위로 bio를 사용하는데, 기본적으로 bio 구조체는 입출력을 수행할 저장장치의 영역 정보와 입출력을 수행할 데이터를 저장하기 위한 메모리 영역의 정보를 포함할 수 있다. 저장장치의 영역 정보는 sector_t 단위로 지정되며, 이 값은 대부분의 저장장치들의 최소 입출력 단위인 512 Byte로 고정되어 있고, 입출력을 수행할 메모리 영역의 정보는 페이지 단위로 지정되며, 이 때 페이지는 보통 4096 Byte라 할 수 있다.

[0064] 또한, 블록 레이어는 입출력의 우선순위에 대한 조절을 위하여 입출력 스케줄러를 포함하고 있으며, 이 입출력 스케줄러는 하드웨어 저장장치로 요청을 보내기 전에 입출력 요청들을 합치거나 정렬하여 더 높은 입출력 성능을 낼 수 있도록 할 수 있다.

[0065] (5) 입출력 스케줄러(IO Scheduler)

[0066] 리눅스 커널은 기본적으로 Noop 스케줄러, Deadline 스케줄러, CFQ(Completely Fair Queuing), 3개의 입출력 스케줄러(IO Scheduler)를 제공할 수 있다.

[0067] Noop 스케줄러는 가장 단순한 스케줄러로, FIFO(First In First Out) 큐로 들어오는 요청을 처리할 수 있다. 이 스케줄러는 가장 낮은 스케줄링 지연을 제공하며 요청을 재정렬 하지 않기 때문에 고성능의 하드웨어 컨트롤러를 장착한 저장장치에 사용하기에 적합할 수 있다.

[0068] Deadline 스케줄러는 입출력 요청에 데드라인을 할당하고, 데드라인이 가까운 요청부터 처리하며, 입출력의 처리량보다 지연시간 감소에 최적화된 스케줄러이며, 일반적으로 낮은 지연시간을 필요로 하는 실시간 시스템에 적합할 수 있다.

[0069] CFQ 스케줄러는 프로세스별 큐를 만들어 입출력 요청을 처리하며, 모든 프로세스가 공평하게 입출력을 처리할 수 있도록 처리시간을 할당하여 다수의 프로세스에서 입출력 요청이 발생하는 상황에서 좋은 성능을 가질 수 있다. 대다수의 운영환경은 다수의 프로세스가 입출력 요청을 발생시키기 때문에 리눅스 커널에서는 CFQ 스케줄러를 기본 입출력 스케줄러로 사용하고 있다.

[0071] 도 2는 본 발명의 실시예로서, 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템을 나타낸 구성도이다.

[0072] 도 2를 참조하면, 본 발명의 실시예에 따른 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템은 상황정보 수집부(100), 상황정보 가공부(200), 상황정보 공유부(300), 입출력 수행부(400)를 포함할 수 있다.

[0073] 상황정보 수집부(100)는 사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집할 수 있다.

[0074] 이 때 상황정보 수집부(100)에서 수집되는 상황정보는 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 접근 횟수, 사용 시간에 기초한 사용빈도 점수를 포함할 수 있다.

[0075] 상기 사용빈도 점수는 하기 수학식 1로 표현되는 애플리케이션 사용빈도 점수 산출식을 이용하여 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 사용빈도를 산출할 수 있다.

[0076] <수학식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid})} + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}$$

[0077]

[0078] (uid : 애플리케이션의 ID, $AUScore_{uid}$: 애플리케이션 uid 의 사용빈도 점수, $WeeklyOpenCount_{ui}$: 7일 간 애플리케이션 uid 의 접근 횟수들의 집합, $WeeklyFGtime_{uid}$: 7일 간 애플리케이션 uid 의 사용 시간들의 집합, $OpenCount_{uid}$: 일별 애플리케이션 uid 의 접근 횟수, $FGtime_{uid}$: 일별 애플리케이션 uid 의 사용 시간)

[0079] 상황정보 가공부(200)는 상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일을 순차적으로 애플리케이션 저장소에 기록하여 상기 상황정보를 가공할 수 있다.

[0080] 보다 상세하게, 상황정보 가공부(200)는 캐시 우선순위 분류부(210), 파일 기록부(230)를 포함할 수 있다.

[0081] 캐시 우선순위 분류부(210)는 상기 수집된 상황정보에 기초한 캐시 적중률 점수에 따라 상기 파일에 대한 캐시 우선순위를 분류할 수 있다.

[0082] 캐시 우선순위 분류부(210)는 상기 캐시 적중률 점수가 상위 1/3 이상이면 높은 우선순위 캐시(High priority cache)로 분류하고, 상기 캐시 적중률 점수가 상위 2/3 이상 1/3 미만이면 표준 우선순위 캐시(Normal priority cache)로 분류하고, 상기 캐시 적중률 점수가 2/3 미만이면 낮은 우선순위 캐시(Low priority cache)로 분류할 수 있다.

[0083] 상기 캐시 적중률 점수는 하기 수학식 2로 표현되는 캐시 적중률 점수 산출식을 이용하여 상기 수집된 상황정보에 기초한 상기 파일에 대한 캐시 적중률 점수를 산출할 수 있다.

[0084] <수학식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

[0085]

[0086] ($inode$: 파일의 inode ID, $CHScore_{inode}$: 캐시 적중률 점수, $ReadCacheHit$: 읽기 캐시 적중 수의 집합, $WriteCacheHit$: 쓰기 캐시 적중 수의 집합, $ReadCacheHit_{inode}$: 파일 inode의 읽기 캐시 적중 수, $WritecacheHit_{inode}$: 파일 inode의 쓰기 캐시 적중 수)

[0087] 파일 기록부(230)는 상기 분류된 캐시 우선순위 중 기 설정된 우선순위 캐시에 해당하는 파일을 상기 애플리케이션 저장소에 기록할 수 있다.

[0088] 상황정보 공유부(300)는 상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행할 수 있다.

[0089] 입출력 수행부(400)는 상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행할 수 있다.

[0090] 한편, 도 2에 도시된 바와 같이 본 발명의 실시예에 따른 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템은 캐시 관리부(510), 미리읽기 수행부(530)를 더 포함할 수 있다.

[0091] 캐시 관리부(510)는 상기 높은 우선순위 캐시는 우선순위 큐(Priority queue)로 관리하고, 상기 표준 우선순위 캐시는 LRU 큐로 관리하고, 상기 낮은 우선순위 캐시는 FIFO 큐로 관리할 수 있다.

[0092] 미리읽기 수행부(530)는 상기 높은 우선순위 캐시(High priority cache)에 해당하는 파일을 단일파일과 다중파일로 구분하여 미리읽기(Readahead)를 각각 수행할 수 있다.

[0094] 도 3은 본 발명의 실시예로서, 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법을 나타낸 순서도이다.

[0095] 도 3을 참조하면, 본 발명의 일 실시예에 따른 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 방법은 상황정보 수집단계(S100), 상황정보 가공단계(S200), 상황정보 공유단계(S300), 입출력 수행단계(S400)를 포함할 수 있다.

[0096] 상황정보 수집단계는 상황정보 수집부(100)에서 사용자 식별자(UID: User identifier)를 기준으로 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 파일의 상황정보를 수집할 수 있다(S100).

[0097] 상황정보 수집단계(S100)에서 수집되는 상황정보는 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 접근 횟수, 사용 시간에 기초한 사용빈도 점수를 포함할 수 있다.

[0098] 사용빈도 점수는 하기 수학적 식 1로 표현되는 애플리케이션 사용빈도 점수 산출식을 이용하여 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션의 사용빈도를 산출할 수 있다.

[0099] <수학식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid})} + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}$$

[0100]

[0101] (uid: 애플리케이션의 ID, $AUScore_{uid}$: 애플리케이션 uid의 사용빈도 점수, $WeeklyOpenCount_{ui}$: 7일간 애플리케이션 uid의 접근 횟수들의 집합, $WeeklyFGtime_{uid}$: 7일 간 애플리케이션 uid의 사용 시간들의 집합, $OpenCount_{uid}$: 일별 애플리케이션 uid의 접근 횟수, $FGtime_{uid}$: 일별 애플리케이션 uid의 사용 시간)

[0102] 상황정보 가공단계는 상황정보 가공부(200)에서 상기 수집된 상황정보를 기준으로 상기 파일의 캐시 우선순위를 분석하고, 상기 파일을 순차적으로 애플리케이션 저장소에 기록하여 상기 상황정보를 가공할 수 있다(S200).

[0103] 보다 상세하게, 상황정보 가공단계(S200)는 캐시 우선순위 분류단계(S210), 파일 기록단계(S230)를 포함할 수 있다.

[0104] 캐시 우선순위 분류단계는 캐시 우선순위 분류부(210)에서 상기 수집된 상황정보에 기초한 캐시 적중률 점수에 따라 상기 파일에 대한 캐시 우선순위를 분류할 수 있다(S210).

[0105] 캐시 적중률 점수는 하기 수학적 식 2로 표현되는 캐시 적중률 점수 산출식을 이용하여 상기 수집된 상황정보에 기초한 상기 파일에 대한 캐시 적중률 점수를 산출할 수 있다.

[0106] <수학식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

[0107]

[0108] (inode: 파일의 inode ID, $CHScore_{inode}$: 캐시 적중률 점수, $ReadCacheHit$: 읽기 캐시 적중 수의 집합, $WriteCacheHit$: 쓰기 캐시 적중 수의 집합, $ReadCacheHit_{inode}$: 파일 inode의 읽기 캐시 적중 수, $WritecacheHit_{inode}$: 파일 inode의 쓰기 캐시 적중 수)

[0109] 캐시 우선순위 분류단계(S210)는 상기 캐시 적중률 점수가 상위 1/3 이상이면 높은 우선순위 캐시(High priority cache)로 분류하고, 상기 캐시 적중률 점수가 상위 2/3 이상 1/3 미만이면 표준 우선순위 캐시(Normal priority cache)로 분류하고, 상기 캐시 적중률 점수가 2/3 미만이면 낮은 우선순위 캐시(Low priority cache)로 분류한다.

e)로 분류할 수 있다.

- [0110] 그리고 상기 분류된 높은 우선순위 캐시, 표준 우선순위 캐시, 낮은 우선순위 캐시는 캐시 관리부(510)에서 각각 관리할 수 있다. 캐시 관리부(510)에서 상기 높은 우선순위 캐시는 우선순위 큐(Priority queue)로 관리하고, 상기 표준 우선순위 캐시는 LRU 큐로 관리하고, 상기 낮은 우선순위 캐시는 FIFO 큐로 관리할 수 있다.
- [0111] 미리 읽기 수행부(530)에서는 상기 높은 우선순위 캐시(High priority cache)에 해당하는 파일을 단일파일과 다중파일로 구분하여 미리읽기(Readahead)를 각각 수행할 수 있다.
- [0112] 파일 기록단계는 파일 기록부(230)에서 상기 분류된 캐시 우선순위 중 기 설정된 우선순위 캐시에 해당하는 파일을 상기 애플리케이션 저장소에 기록할 수 있다(S230).
- [0113] 상황정보 공유단계는 상황정보 공유부(300)에서 상기 가공된 상황정보를 리눅스 커널로 공유하거나, 상기 리눅스 커널 내부 레이어 간 공유를 수행할 수 있다(S300).
- [0114] 입출력 수행단계는 입출력 수행부(400)에서 상기 공유된 상황정보를 기준으로 상기 사용빈도가 높은 적어도 하나 이상의 애플리케이션에 대한 입출력 스케줄링을 수행한 후, 상기 수행된 입출력 스케줄링에 따라 입출력을 수행할 수 있다(S400).
- [0116] 도 4는 본 발명의 실시예에 따라 본 발명의 시스템을 안드로이드 운영체제에 적용한 구성도를 나타낸 도면이다.
- [0117] 도 4를 참조하면, 본 발명의 실시예에 따라 본 발명의 시스템을 안드로이드 운영체제에 적용한 구성도는 Context Analysis 모듈, VFS/FD Mapper, Context-aware Readahead, Context-ware Cache Queue, Context-ware Defragger, CFQ-CA를 포함할 수 있다.
- [0118] 먼저, Context Analysis 모듈은 안드로이드 운영체제에서 애플리케이션을 구동하는 Android Framework 레이어에서 애플리케이션 별 정보 및 리눅스 커널 입출력 스택에서 생성된 정보의 수집과 분석을 통해, 하위 모듈들이 활용할 수 있는 상황인지 정보를 생성할 수 있다.
- [0119] Context Analysis 모듈은 상황정보 수집부에 해당하는 I/O Context Manager(100), 상황정보 가공부에 해당하는 I/O Context Analyzer(200)를 포함할 수 있다.
- [0120] (1) I/O Context Manager(100)
- [0121] I/O Context Manager(100)는 본 발명의 시스템이 적용된 안드로이드 운영체제의 하위 모듈들에서 사용할 수 있는 애플리케이션 사용통계, 파일 사용통계 등의 상황정보들을 수집하고 관리할 수 있다. I/O Context Manager(100)에서는 애플리케이션 단위 별 상황정보를 수집할 수 있다.
- [0122] 모든 애플리케이션의 파일 사용통계를 지속적으로 수집하고 분석하는 것은 시스템 오버헤드가 크며, 스마트폰에서의 12% 정도의 애플리케이션이 전체 애플리케이션 사용시간의 50% 이상을 차지하고, 65% 정도의 애플리케이션이 전체 애플리케이션 실행 빈도의 80% 이상을 보이는 특성을 고려하였을 때, 모든 애플리케이션의 파일 사용통계를 수집하는 것은 비효율적이라 할 수 있다.
- [0123] 이에 사용자가 자주 사용하는 애플리케이션에 대해서만 파일 사용통계를 수집하기 위해서 안드로이드 시스템에서 애플리케이션을 구별하는 단위인 uid를 기준으로 애플리케이션 사용통계를 수집한 후, 파일 사용통계 컨텍스트(상황정보)를 수집할 대상 애플리케이션을 선정할 수 있다.
- [0124] 대상 애플리케이션 선정을 위해, I/O Context Manager(100)에서는 애플리케이션 접근횟수 및 사용시간을 측정할 수 있다. 스마트폰에서 애플리케이션 사용패턴에 따르면, 일간 사용통계가 사용자가 주로 사용하는 애플리케이션을 파악하기 위한 최소단위로써 적합하고, 주간 사용통계가 주로 사용하는 애플리케이션의 변화를 파악하기 위한 단위로 적합할 수 있다. 이에 일별 애플리케이션 접근횟수 및 사용시간을 측정하여 기록을 하고, 이전 7일간의 접근횟수 및 사용시간의 평균값으로 주별 애플리케이션 접근횟수 및 사용시간을 생성할 수 있다.
- [0125] 생성된 애플리케이션 접근횟수 및 사용시간은 해당 애플리케이션의 사용빈도 점수를 추정하는 데에 사용되며, 애플리케이션의 사용빈도 점수는 하기 수학적 식 1을 통해 산출할 수 있다. 이 때 사용되는 변수들은 아래 표 1과 같다.

[0126] <수학식 1>

$$AUScore_{uid} = \frac{OpenCount_{uid}}{\max(WeeklyOpenCount_{uid}) + \frac{FGtime_{uid}}{\max(WeeklyFGtime_{uid})}}$$

[0127]

[0128] <표 1>

파라미터	정의
<i>uid</i>	애플리케이션의 ID
<i>AUScore_{uid}</i>	애플리케이션 <i>uid</i> 의 사용빈도 점수
<i>WeeklyOpenCount_{ui}</i>	7일간 애플리케이션 <i>uid</i> 의 접근 횟수들의 집합
<i>WeeklyFGtime_{uid}</i>	7일간 애플리케이션 <i>uid</i> 의 사용 시간들의 집합
<i>OpenCount_{uid}</i>	일별 애플리케이션 <i>uid</i> 의 접근 횟수
<i>FGtime_{uid}</i>	일별 애플리케이션 <i>uid</i> 의 접근 횟수

[0129]

[0130] 각 애플리케이션마다 구해진 사용빈도 점수를 기반으로 상위 N개의 애플리케이션에 대해 파일 사용통계 컨텍스트를 수집 및 저장하여, 해당 애플리케이션들의 입출력 요청에 대해서 CAFIO 모듈들이 동작하여 입출력 성능을 향상시킬 수 있도록 할 수 있다.

[0131]

I/O Context Manager(100)에서 수집하는 파일 사용 통계는 애플리케이션 별 사용되는 각 파일들의 사용빈도와 두 개 이상의 파일 간 접근 패턴, 이 두 가지를 수집할 수 있다. 애플리케이션 별 파일의 사용빈도는 특정 파일이 얼마나 많이 사용되는지를 나타내며, 수집된 값에 따라 Hot, Warm, Cold 세 가지로 분류하여 관리할 수 있다. 그리고 애플리케이션 별 파일 간 접근 패턴은 해당 애플리케이션이 파일을 읽은 순서를 기록하며, 사용빈도에 따라 분류가 Hot인 파일들에 대해서만 추적을 하도록 하여 시스템 오버헤드를 줄일 수 있도록 할 수 있다.

[0132]

애플리케이션 별 파일의 사용빈도는 하기 알고리즘 1에 따라 수집되며, 파일시스템 상 파일의 Inode ID, VFS단에서의 읽기/쓰기 횟수, 파일 열기 횟수를 수집할 수 있다.

[0133] <알고리즘 1>

```

1 Procedure onFileOpen_FileUsage( inodeid )
2   stat = getFileUsageStatisticByInode(inodeid)
3   stat.FileReadCount++
4   stat.FileWriteCount++
5 End Procedure
6
7 Procedure onFileRead_FileUsage( inodeid )
8   stat = getFileUsageStatisticByInode(inodeid)
9   If( stat.lastReadTimestamp + Timeout < CurrentTimestamp )
10    stat.FileReadCount++
11   End if
12   stat.lastReadTimestamp = CurrentTimestamp
13 End Procedure
14
15 Procedure onFileWrite_FileUsage( inodeid )
16   stat = getFileUsageStatisticByInode(inodeid)
17   If( stat.lastWriteTimestamp + Timeout < CurrentTimestamp )
18    stat.FileWriteCount++
19   End if
20   stat.lastReadTimestamp = CurrentTimestamp
21 End Procedure

```

[0134]

[0135] 상기 알고리즘 1의 onFileOpen은 Linux커널의 VFS 레이어에서 파일 사용통계 수집 대상인 애플리케이션이 파일을 열 때 호출되며, 해당 파일에 대한 통계를 inode를 통해 얻어온 다음 Read/Write 카운트를 1씩 늘려주는 역

할을 할 수 있다.

[0136] onFileRead는 마찬가지로 Linux커널의 VFS에서 파일 사용 통계 수집 대상인 애플리케이션이 파일을 읽을 때 호출되며, 해당 파일에 대한 통계를 inode를 통해 얻어온 다음 Read 카운트를 1씩 늘려주는 역할을 할 수 있다.

[0137] 이 때, 파일을 마지막으로 읽은 시점에서 Timeout만큼의 시간이 지나지 않았다면 Read 카운트를 늘리지 않을 수 있다. 이는 일반적으로 파일의 크기가 큰 경우 연속적인 읽기가 많이 발생할 수 있는데, 이 경우 모든 읽기 작업을 접근횟수로 카운트할 때 파일의 용량에 비례하여 Read 카운트가 늘어나기 때문에 실제 파일의 사용빈도를 측정하는데 문제가 되기 때문이다.

[0138] 마찬가지로 onFileWrite에서도 onFileRead와 동일하게 동작하며, Linux커널의 VFS에서 파일 사용 통계 수집 대상인 애플리케이션이 파일을 쓸 때 호출될 수 있다.

[0139] (2) I/O Context Analyzer(200)

[0140] I/O Context Analyzer(200)는 I/O Context Manager(100)에서 수집한 정보를 활용하여 본 발명의 시스템이 적용된 안드로이드 운영체제의 다른 모듈들에서 사용할 수 있도록 정보를 가공할 수 있다.

[0141] 이 때 I/O Context Analyzer(200)는 캐시 관리부에 해당하는 Context-aware Cache Queue(510)에서 사용하기 위한 파일별 캐시 우선순위를 분석 및 분류하는 캐시 우선순위 분류부(210)를 포함할 수 있다.

[0142] 캐시 우선순위 분류부(210)의 파일별 캐시 우선순위는 I/O Context Manager(100)에서 수집된 파일의 사용빈도를 이용하여 분류를 하며, 이의 분류방법은 아래 표 2를 참조할 수 있다.

[0143] <표 2>

분류	조건
High priority cache	Read Hot or Write Hot
Normal priority cache	Read Warm or Write Warm
Low priority cache	Read Cold and Write Cold

[0144]

[0145] 이 때, High priority cache 그룹은 우선순위를 구하기 위하여 각 파일의 캐시 적중 횟수를 분석하여 읽기 캐시 적중 수와 쓰기 캐시 적중 수를 추적할 수 있다. 읽기 캐시 적중 수는 읽기 캐시가 적중된 페이지수를 추적하도록 하며, 쓰기 캐시 적중 수는 읽기-수정-쓰기가 수행된 횟수를 추적하며, 쓰기의 크기가 파일시스템의 블록사이즈보다 작은 경우도 추적하도록 할 수 있다.

[0146] 추적된 읽기 캐시 적중 수와 쓰기 캐시 적중 수는 매 시간마다 적중률을 산출하기 위해 계산되거나 읽기, 쓰기 캐시 적중 수를 저장하고 있는 변수가 최대값에 도달했을 때 수행될 수 있다.

[0147] 캐시 적중률을 정규화한 캐시 적중률 점수는 하기 수학식 2를 통하여 산출할 수 있으며, 이 때 사용되는 변수들은 아래 표 3을 참조할 수 있다.

[0148] <수학식 2>

$$CHScore_{inode} = \frac{ReadCacheHit_{inode}}{\max(ReadCacheHit)} + \frac{WriteCacheHit_{inode}}{\max(WriteCacheHit)}$$

[0149]

[0150] <표 3>

파라미터	정의
<i>inode</i>	파일의 inode ID
<i>CHScore_{inode}</i>	캐시 적중률 점수
<i>ReadCacheHit</i>	읽기 캐시 적중 수의 집합
<i>WriteCacheHit</i>	쓰기 캐시 적중 수의 집합
<i>ReadCacheHit_{inode}</i>	파일 <i>inode</i> 의 읽기 캐시 적중 수
<i>WriteCacheHit_{inode}</i>	파일 <i>inode</i> 의 쓰기 캐시 적중 수

[0151]

[0152] 또한, I/O Context Analyzer(200)는 미리읽기 수행부에 해당하는 Context-aware Readahead(530)에서 사용하기 위한 파일의 순차 접근 패턴을 분석 및 기록하는 파일 기록부(230)를 포함할 수 있다.

[0153] 파일 기록부(230)의 파일 순차 접근 패턴은 I/O Context Manager(100)에서 수집된 파일의 사용빈도를 이용하여, 파일의 사용빈도가 Hot인 파일을 대상으로 순차적으로 읽힌 파일들의 순서를 기록할 수 있다. 이는 추후 Context-aware Readahead(530)에서 해당 기록을 사용할 수 있도록 하며, 하기 알고리즘 2에 따라 분석되어 분류되게 할 수 있다.

[0154] <알고리즘 2>

```

1 Procedure onFileRead( inodeid )
2   stat = getFileUsageStatisticByInode(inodeid)
3   IF isFileHot( inodeid )
4     If( ( stat.lastReadInodeid not inodeid ) and
5         ( stat.lastReadTimestamp + Timeout < CurrentTimestamp)
6     and
7         ( getFileSize( inodeid ) < Treadahead ) )
8       trace = getTraceOfFile( inodeid )
9       trace.insertAfter( stat.lastReadInodeid, inodeid )
10    End if
11  End if
12  stat.lastReadTimestamp = CurrentTimestamp
13  stat.lastReadInodeid = inodeid
End Procedure

```

[0155]

[0156] 상기 알고리즘 2의 onFileOpen은 Linux커널의 VFS에서 파일 사용통계 수집 대상인 애플리케이션이 파일을 열 때 호출되며, 파일의 순차 접근 패턴을 분석하도록 할 수 있다.

[0157] 파일의 순차 접근 패턴 분석의 오버헤드를 줄이고 불필요한 통계를 수집하는 것을 막기 위하여 Line 3에서 해당 파일이 Hot 파일인지 확인하여 Hot 파일일 때만 통계를 수집하도록 할 수 있다.

[0158] 그리고 Line 4-6에서 파일 읽기의 순차성을 확인하기 위하여 마지막으로 읽은 파일이 다시 읽혔는지 확인하여 마지막으로 읽은 파일이 다시 읽힌 경우에는 무시하도록 할 수 있다.

[0159] 또한 마지막으로 읽은 시점이 일정 시간 Timeout을 넘겼는지 확인하여 넘겼을 경우 무시하도록 하며, 마지막으로 읽기가 요청된 파일의 크기가 Readahead크기보다 큰지 확인하여 클 경우 무시하도록 할 수 있다.

[0160] 파일의 순차성을 확인한 결과 파일이 순차적으로 읽혔다고 판단된 경우, Line 8에서 현재 읽기 요청된 파일을 마지막으로 읽은 파일 뒤로 오도록 하여 순서를 보존하여 추후 Context-aware Readahead 모듈에서 사용할 수 있도록 할 수 있다.

[0161] (3) VFS/FD Mapper(300)

[0162] 상황정보 공유부에 해당하는 VFS/FD Mapper(300)는 Android Framework 레이어와 리눅스 커널 입출력 스택 간 정보 공유, 및 리눅스 커널 입출력 스택 내부 레이어 간 상황인지 정보 공유를 수행할 수 있다.

[0163] VFS/FD Mapper(300)는 Android Framework와 리눅스 커널 내부에서 처리되는 입출력을 추적하기 위하여 각 애플리케이션 Process들의 File descriptor와 리눅스 커널의 VFS entry를 매핑하고, IO tag를 생성하여 리눅스 커널의 각 계층에 존재하는 CAFIO 모듈들이 서로 효율적으로 정보를 전달할 수 있도록 할 수 있다.

[0164] (4) CFQ-CA(400)

[0165] 입출력 수행부에 해당하는 CFQ-CA(400)는 블록 레이어(Block Layer)의 입출력 스케줄러(IO Scheduler)에서 동작하며, 사용자가 현재 사용하고 있는 애플리케이션의 입출력 요청을 우선적으로 처리함으로써, 사용자 요청에 대한 입출력 지연을 최소화할 수 있다.

[0166] CFQ-CA(400)은 리눅스의 입출력 스케줄러를 대체하여 동작을 하게 되며, 사용자가 현재 사용하고 있는 애플리케이션의 입출력을 우선적으로 처리하기 위한 입출력 스케줄러라 할 수 있다. 이는 I/O Context Analyzer(200)에서 분석된 (입출력 우선순위(I/O Priority)를 기반으로 입출력 스케줄링(I/O Scheduling)을 처리하여, 사용자가 현재 사용하고 있는 애플리케이션 및 해당 애플리케이션의 서비스들이 보다 높은 우선순위로 처리될 수 있도록 할 수 있다. 이는 사용자가 당장 필요로 하는 입출력작업들의 지연시간을 줄여 더 나은 사용자 경험을 제공할

수 있도록 할 수 있다.

[0167] (5) Context-aware Cache Queue(510)

[0168] 캐시 관리부에 해당하는 Context-aware Cache Queue(510)는 상황인지 정보를 활용하여 캐시의 히트율을 높이고, 불필요한 캐시를 줄여 전체 캐시 활용율을 높일 수 있다.

[0169] Context-aware Cache Queue(510)은 캐시된 파일들의 효율적인 관리를 위하여, 앞서 I/O Context Analyzer(200)에서 분석된 캐시 우선순위에 따라 파일 페이지 캐시를 관리하는 역할을 하는 모듈이라 할 수 있다. High priority cache에 대해서는 해당 파일의 캐시 우선순위에 따른 Priority queue로 관리를 하고, Normal priority cache에 대해서는 LRU queue로 관리를 하며, Low priority cache에 대해서는 FIFO queue로 관리를 할 수 있다. 그리고 이 세 가지의 queue들을 priority queue로 통합 관리를 하여, 가장 먼저 Low priority cache 부터 evict되도록 관리를 하여 재사용율이 높은 파일 캐시들이 최대한 메인 메모리에 머물러 있을 수 있도록 할 수 있다.

[0170] (6) Context-ware Readahead(530)

[0171] 미리읽기 수행부에 해당하는 Context-ware Readahead(530)는 리눅스 커널의 페이지 캐시에서 동작하며, 상황인지 정보를 활용한 미리읽기를 수행하여 추후 필요한 데이터를 미리 읽는 것에 따른 효율성을 높일 수 있다.

[0172] Context-ware Readahead(530)는 파일의 캐시 적중률을 높이기 위하여 앞으로 필요할 수 있는 파일을 미리 메모리로 읽어오는 역할을 하는 모듈로써, 단일 파일 Readahead와 다중 파일 Readahead를 수행할 수 있다. 단일 파일 Readahead는 하나의 파일을 대상으로 파일의 일부 혹은 전체를 읽어오는 것으로, 대상 파일이 Readahead 크기(512KB)보다 크며, Sequential Read로 분류되었을 경우, 현재 읽기 요청 이후 Readahead 크기만큼 해당 파일의 내용을 미리 읽어 캐시 적중률을 높일 수 있도록 할 수 있다. 다중 파일 Readahead는 다수의 파일을 대상으로 파일의 일부 혹은 전체를 읽어오는 것으로, 미리 파악된 파일의 순차 접근 패턴을 사용하여 해당 접근 패턴에 해당하는 파일의 읽기 요청이 들어왔을 경우, 해당 패턴에 정의된 파일 순서대로 파일들을 Readahead 크기만큼 미리 읽어 캐시 적중률을 높일 수 있도록 할 수 있다.

[0173] (7) Context-ware Defragger

[0174] 한편, 도 4에 도시된 Context-ware Defragger은 Ex4 File System에서 동작하며, 조각난 데이터를 모아 데이터를 효율적으로 한꺼번에 전달하기 쉽게 관리할 수 있다.

[0175] Context-aware Defragger는 Context-ware Readahead(530)에서 다중 파일 Readahead에서 사용되는 연속되어 자주 쓰이는 작은 크기의 파일들을 하위 저장장치의 연속된 공간에 저장될 수 있도록 하여 해당 파일들의 읽기 성능을 높이기 위한 모듈이라 할 수 있다. 이 모듈은 Bulk transfer(512KB)를 이용하여 해당 데이터가 하부 저장장치가 자체 FTL을 가지고 있는 Flash 기반 저장장치이더라도 최대한 연속된 공간에 저장되어 추후 해당 데이터를 읽어올 때 빠르게 읽어올 수 있도록 할 수 있다. 이 Bulk transfer를 이용한 파일의 relocation은 파일을 다시 쓰는 작업을 수행하기 때문에 파일의 내용 변화가 없으며, 최대한의 성능향상을 기대할 수 있는 다중 파일 Readahead 대상 파일들 중 Write가 없는 파일들을 대상으로만 수행할 수 있다.

[0177] 도 5는 본 발명의 실시예에 따른 본 발명의 성능 평가 중, 애플리케이션 실행시간 테스트 결과를 나타낸 도면이고, 도 6은 본 발명의 실시예에 따른 본 발명의 성능 평가 중, 캐시 적중률 테스트 결과를 나타낸 도면이고, 도 7은 본 발명의 실시예에 따른 본 발명의 성능 평가 중, 읽기 속도 테스트 결과를 나타낸 도면이다.

[0178] 본 발명인 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템 및 방법의 성능을 평가하기 위해, 애플리케이션의 실행 시간(Launch time), 캐시 적중률(Cache ratio), 읽기 속도(Read speed)를 측정하여 기존의 안드로이드 시스템과 비교를 수행하였다. 기존의 안드로이드 시스템은 파란색의 Android이며, 본 발명의 시스템은 주황색의 Android+CAFIO(Context Aware I/O Management System)이다. 테스트가 진행된 디바이스의 사양은 하기 표 4와 같다.

[0179] <표 4>

프로세서	Qualcomm Snapdragon 810 MSM8994 SoC
메모리	3GB LPDDR4 SDRAM
디스플레이	WQHD AMOLED Touchscreen Display
저장장치	32GB Flash Storage(eMMC 5.0), MLC
운영체제	Android 7.1.2 (Nougat) with Linux Kernel 3.10.73

[0180]

[0181] 또한, 성능 측정을 위해, Social, Multimedia, Game, Productivity 카테고리의 인기 애플리케이션을 각 5개씩, 총 20개의 애플리케이션을 대상으로 테스트를 진행하였다. 선정 한 애플리케이션은 하기 표 5와 같다.

[0182] <표 5>

기호	애플리케이션	기호	애플리케이션
A	밴드	X	리니지 M
B	Instagram	L	헬로히어로에픽배틀
C	Faceook	M	프렌즈캠
D	카카오톡	N	영원한 7일의 도시
E	네이버 카페	O	포켓몬케스트
F	TikTok	P	한컴오피스 Viewer
G	okusu	Q	네이버 클라우드
H	SBS - 온에어	R	플라리스 오피스
I	멜론	S	Microsoft Powerpoint
J	뮤직메이트	T	Microsoft Word

[0183]

[0184] 모든 테스트는 상기 표 5의 애플리케이션들을 한 번씩 순서대로 실행한 것을 1회의 테스트로 간주하며, 각 테스트 간 명시적인 애플리케이션 종료 및 캐시 invalidation을 하여 테스트 간 캐시가 결과에 영향을 주지 않도록 하였다.

[0185] 테스트 방법으로는, 애플리케이션의 실행 시간(Launch time)의 경우 애플리케이션을 실행하기 시작한 순간부터 애플리케이션의 메인 Activity가 화면에 보여지기 까지의 시간을 측정한 것이며, 캐시 적중률은 애플리케이션의 실행 시간을 측정하는 동안 발생한 해당 애플리케이션에서 발생한 모든 읽기 요청의 캐시 적중률이며, 읽기 속도는 애플리케이션의 실행 시간을 측정하는 동안 발생한 해당 애플리케이션에서 발생한 모든 읽기 요청의 읽기 속도의 평균값이다.

[0186] 그 결과, 도 5에 도시된 바와 같이 애플리케이션 실행 시간 테스트 결과를 확인할 수 있다.

[0187] 도 5를 참조하면, 테스트 대상 20개의 애플리케이션에서 평균 7%의 성능향상을 확인할 수 있다. 특히 애플리케이션 K(리니지 M)의 경우는 47%의 성능 향상을 보이며 15.9 초에서 10.8초로 5.1초의 실행시간 감소 효과를 보였다. 이는 다수의 작은 파일을 읽어 들이는 해당 애플리케이션의 특성상 캐싱 정책과 잘 맞아 성능향상을 보인 것이다.

[0188] 또한, 도 6에 도시된 바와 같이 캐시 적중률 테스트 결과를 확인할 수 있다.

[0189] 도 6을 참조하면, 테스트 대상 20개의 애플리케이션에서 평균 11%의 성능향상을 확인할 수 있다. 상기 애플리케이션 실행시간 테스트와 비슷하게, 애플리케이션 K(리니지 M)의 경우 74%의 성능향상을 보여주었다.

[0190] 도 7에 도시된 바와 같이, 읽기 속도 테스트 결과를 확인할 수 있다.

[0191] 도 7을 참조하면, 테스트 대상 20개의 애플리케이션에서 평균 7배의 성능향상을 확인할 수 있다. 특히 애플리케이션 F(TikTok)의 경우 0.05MB/s 에서 4.42MB/s로 82배의 성능향상을 보였으며, 이는 소수의 작은 파일을 읽어 오는 해당 애플리케이션의 특성상 제안하는 시스템에서 수행하는 Context-aware Defragger와 Context-aware Readahead로 인해 다수의 작은 파일들을 파일 단위가 아닌 연속된 큰 블록단위로 읽어와 I/O Request 수를 줄여 읽기 성능이 비약적으로 늘어났으며, 높은 캐시 적중률로 인해 보조 저장장치가 아닌 주 저장장치에서 바로 읽

어 들었기 때문에 읽기속도가 비약적으로 향상되었다.

[0192] 이와 같이 본 발명인 모바일 기기에서의 상황정보를 기반으로 한 파일 입출력 관리 시스템 및 방법은 모바일 장치에서 사용자의 사용 경험을 높이기 위하여 상황인지 정보를 활용한 애플리케이션 레이어, VFS레이어, 블록 레이어간 정보 전달을 통하여 리눅스 입출력 스택 전반을 개선할 수 있는 효과를 가진다.

[0193] 이는 기존의 안드로이드 시스템에서 발생하는 문제들을 해결하기 위해, 모바일 운영체제에서 실행되는 각 애플리케이션의 파일 사용 패턴을 분석하여 곧 사용할 파일의 일부 혹은 전체를 캐싱하며, 저장장치에서 같이 자주 쓰이는 작은 파일들을 연속적으로 읽어올 수 있도록 조각 모음 하여 읽기 성능을 향상시킬 수 있다.

[0194] 또한 현재 사용 중인 애플리케이션의 입출력 우선순위를 높여 해당 애플리케이션의 입출력 작업을 우선적으로 처리함으로써, 입출력 지연시간을 줄여 애플리케이션의 반응성을 높이고 사용성을 개선할 수 있다.

[0195] 도 5 내지 도 7의 실험 결과, 애플리케이션 실행시간에서는 평균 7%의 성능향상이 있었고, 캐시 적중률에선 평균 11%의 성능향상이, 읽기 속도에서는 평균 7배의 성능향상을 보인 바 있다.

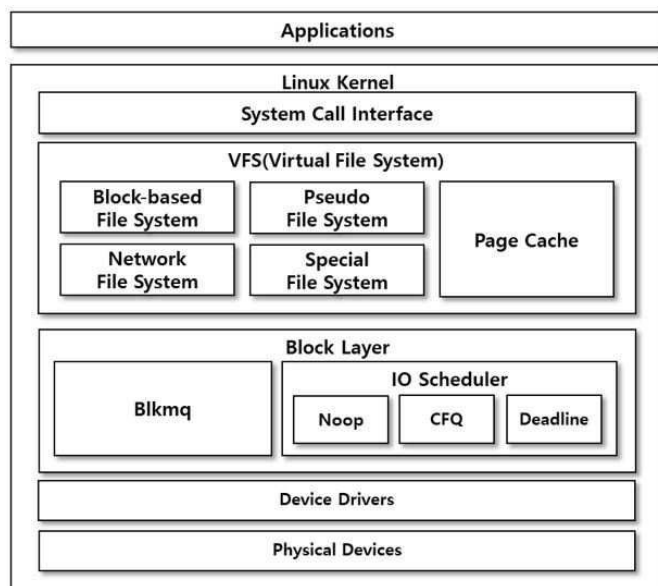
[0196] 이상에서 본 발명의 바람직한 실시예를 설명하였으나, 본 발명은 다양한 변화와 변경 및 균등물을 사용할 수 있다. 본 발명은 상기 실시예를 적절히 변형하여 동일하게 응용할 수 있음이 명확하다. 따라서 상기 기재 내용은 다음 특허청구범위의 한계에 의해 정해지는 본 발명의 범위를 한정하는 것이 아니다.

부호의 설명

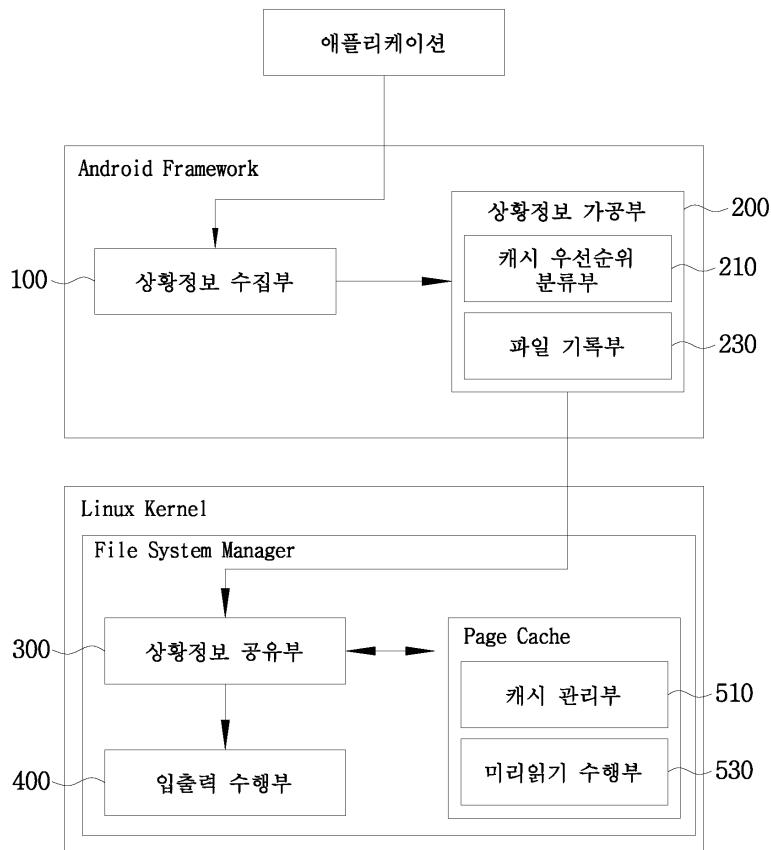
- [0197] 100 : 상황정보 수집부 (I/O Context Manager)
200 : 상황정보 가공부 (I/O Context Analyzer)
210 : 캐시 우선순위 분류부
230 : 파일 기록부
300 : 상황정보 공유부 (VFS/FD Mapper)
400 : 입출력 수행부 (CFQ-CA)
510 : 캐시 관리부 (Context-aware Cache Queue)
530 : 미리읽기 수행부 (Context-aware Readahead)

도면

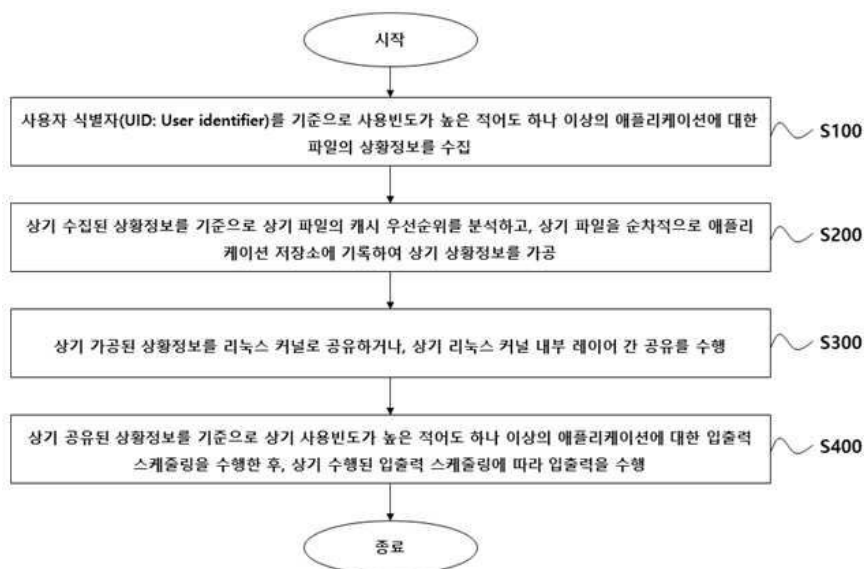
도면1



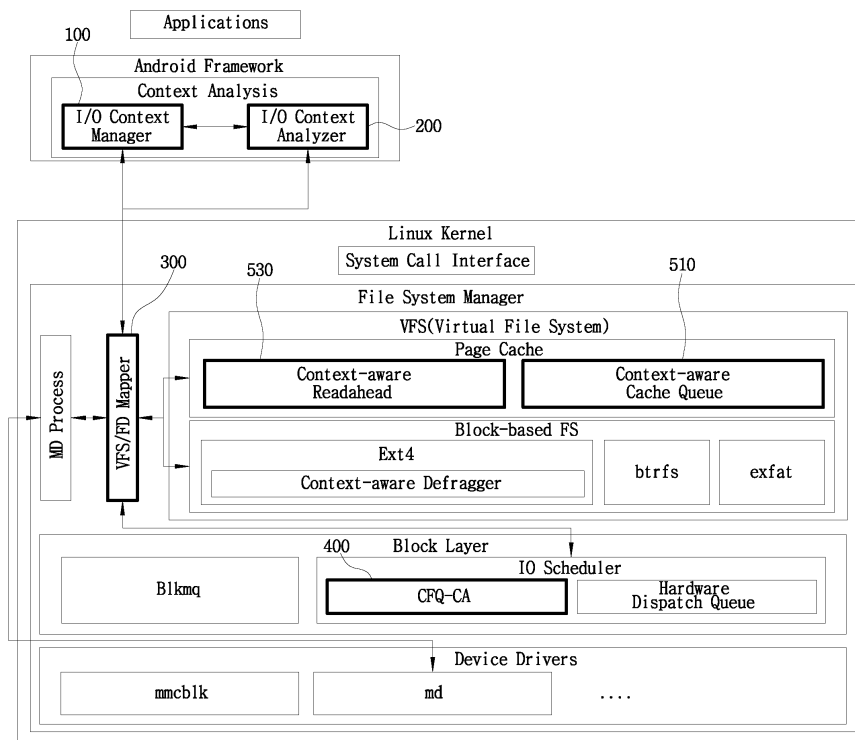
도면2



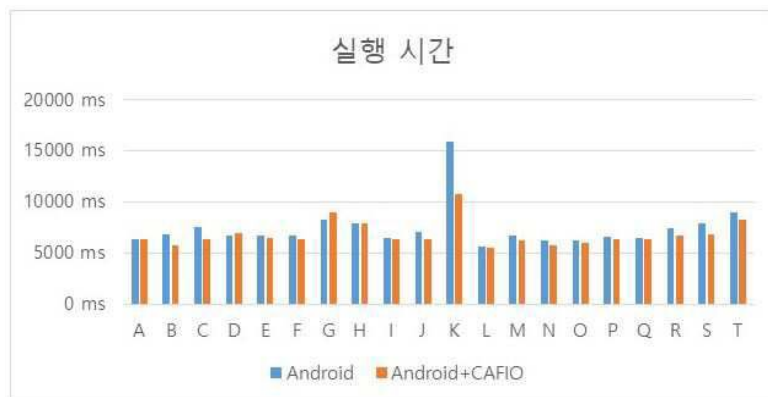
도면3



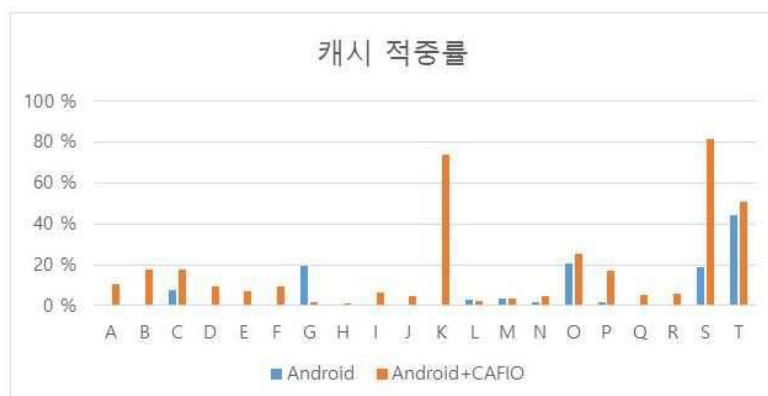
도면4



도면5



도면6



도면7

